最短経路アルゴリズムの平均比較回数について (実験整数論および組合せ理論と計算機)

AUTHOR(S):

野下, 浩平; 増田, 悦夫

CITATION:

野下, 浩平 ...[et al]. 最短経路アルゴリズムの平均比較回数について (実験整数論および組合せ理論と計算機). 数理解析研究所講究録 1977, 301: 118-128

ISSUE DATE:

1977-07

URL:

http://hdl.handle.net/2433/103803

数理解析研究所講究録
301巻 1977年 118-128

最短経路アルゴリズムの平均比較回数について

野下 浩平　　増田 悦夫†
(電気通信大学 計算機科学科)

梗概: 最短経路問題に対するDijkstraアルゴリズムについて,その平均比較回数を計算機実験により調べる. 得られた事実の主要なものは次の通り: 十分大きい完全グラフに対して, 枝の重みが一様分布である時, Dijkstra法の第Δステップにおいて各節点に付与されている距離が新しい値に更新される確率は $1/\Delta$ になる. またその他いくつかの型の分布については, 上記の確率は一様分布のそれよりも大きくない.

これらの事実を利用して, アルゴリズムに要する比較の平均回数の上限を与え, その実現方法を提案する.

同様の事実は, 最短極大木を求めるPrim法についても見られる. ただし, この場合は入力分布の型に依存しない.

これらの実験的事実に対する直観的な解明を与える.

---

† 現在: 日本電信電話公社武蔵野電気通信研究所

On the Expected Behaviors of the Dijkstra's Shortest Path Algorithm for Complete Graphs

Kohei NOSHITA and Etsuo MASUDA
Department of Computer Science
The University of Electro-Communications
Chofu, Tokyo 182, Japan

Summary

Several interesting phenomena observed in the computing experiment on the Dijkstra's shortest path algorithm for complete graphs with non-negatively weighted edges are presented. Those results suggest an expectedly efficient implementation of the algorithm. The similar phenomena are observed in the Prim's shortest spanning tree algorithm. An intuitive elucidation of the underlying grounds for those results is briefly described.

1. Introduction

The purpose of this paper is to report some experimental phenomena observed in the Dijkstra's shortest path algorithm [1] for complete graphs. Those results suggest an expectedly efficient implementation of the algorithm. The similar phenomena are observed in the Prim's shortest spanning tree algorithm [2].

We shall consider the single-source shortest path problem for complete graphs with non-negatively weighted edges [3]. The complexity of the algorithms will be evaluated in terms of binary comparisons between two distances from the source vertex. It is well-known that the Dijkstra's algorithm is most efficient, and in fact optimal, in the sense of the worst case complexity [4]. Certain implementation techniques of this algorithm have been proposed for various types of graphs (e.g., [5],[6],[7],[8]).

In some experiments those techniques have been demonstrated to be effective in reducing the expected number of comparisons. Furthermore they are also effective even for "complete" graphs [8]. However, to the authors' knowledge, the expected behaviors of the algorithm have not yet been investigated in detail. This paper experimentally shows some basic properties of the expected behaviors by counting the number of times of updating the distance, assigned to each vertex in each iterative step of the algorithm.

2. Description of the Algorithm

We shall briefly describe the Dijkstra's algorithm. For general terminologies and previous results, see [3].

Let G be a complete directed graph with set V of n vertices. For each edge (v_i,v_j) a non-negative real number $l(v_i,v_j)$ is initially given as an edge weight, where v_i and v_j are in V such that $v_i \neq v_j$. Let v_0 denote the specified "source" vertex in V. The following Dijkstra's algorithm finds the set of all shortest paths from v_0 to any other vertices in V. In the algorithm the value of D[v] stands for the path-distance assigned tentatively or permanently to vertex v.

```
begin
Initialize:  for each v in V do D[v]:=∞;
     D[v0]:=0;  w:=v0;  P:= {v0} ;
Iterative step:
     for s:=1 step 1 until n-1 do
       begin
       UPDATE:  for each v in V-P do
                  if D[v] > D[w]+l(w,v) then D[v]:=D[w]+l(w,v);
       FINDMIN:  find w such that D[w]=min { D[v] | v in V-P } ;
         P:=P∪{w}
       end
end
```

Note that P denotes the set of vertices each of whose path-distances has been permanently determined.

Throughout the algorithm the number of comparisons required in the UPDATE step is m/2, where m denotes the number of edges ($m=n^2-n$). This is "obligatory" in the algorithm. If the relation in UPDATE (line 8) always holds, it is necessary in the FINDMIN step to find a vertex with the minimum distance out of all the tentative vertices, each of whose distances has been updated to a new smaller value in just the previous UPDATE step. Hence the total number of comparisons necessary in FINDMIN will be $(n-1)*(n-2)/2 \approx n^2/2$ in the worst case. In such cases the set of tentative vertices should be represented in a linear array in order to enable each actual updating within a certain constant time. On the other hand, in the expected case, it is quite natural to expect that the number of updating times will be far less than the number of tentative vertices in V-P in each UPDATE step. Then, as stated in 1, the number of comparisons in FINDMIN can be decreased by employing a priority queue to represent the set V-P of tentative vertices.

3. Counting Results in the Experiment†

3.1. The Number of Updating Times for Uniform Distributions

The first experiment gives the counting results of the total number of updating times in UPDATE throughout the algorithm. See Table 1. The input data initially assigned to edges are taken from the uniform random real numbers in the interval [0,1). It will be worth while to mention that the linear, not square, amount of memory space is sufficient for this experiment, because of the property of random numbers.

In the table each entry is computed as an average value of 10 trials. The parenthesized value is the standard deviation for the corresponding average value.

---

† The experiments have been independently performed by each of the authors. The one is on HITAC 8350 in FORTRAN, while the other on MELCOM 70 in PRIMAL-75. The pseudo-random number generators are due to [9].

In order to examine the more detailed behaviors, the number of updating times in each iterative step of the algorithm has been counted. The average values of 10 trials for n=512 and 2048 are plotted in Figure 2.

Let p(n,s) denote the probability with which, for each vertex in V-P, its value is updated in the s-th UPDATE step for the complete graph of n vertices. In our experiment we observe the following simple phenomena:

"For uniformly distributed input data, the number of vertices whose values are updated may be well approximated to be (n-s)/s. Equivalently p(n,s) will be approximately 1/s for such input data."

Note that the above formula is much closer to the observed results for small s than for large s (near to n).

Let F(n) denote the expected total number of updating times throughout the algorithm for the complete graph of n vertices. The above result leads to the following consequence under the same assumption for input data:

"The simple approximated formula for F(n) will be $\sum_{s=1}^{n-1}(n-s)/s \approx n \log_e n - 0.4228n$, for sufficiently large n."

3.2. An Efficient Implementation

Those experimental results suggest the efficient implementation of the algorithm. For our discussion we shall use a k-ary heap to represent the set V-P of tentative vertices. Then we can estimate the upperbound of the total number of comparisons, denoted by C(n), as follows [6]:

$$C(n)=m/2 + F(n)\ \log_k n + nk \log_k n\ .$$

The first term is obligatory, while the middle term is for the rearrangement of the k-ary heap in each updating. The last term is derived from n deletions of the minimum-valued vertex in the FINDMIN step.

By the above approximation, we can assume that $F(n)=n \log_e n$. In order to minimize C(n) we will choose $\log_e n$ for k. Then C(n) will be expectedly bounded by

$$m/2 + 2n(\log_e n)^2/\log_e \log_e n\ .$$

Thus, we can expect that the number of comparisons required in the algorithm is asymptotically equal to half the number in the worst case, since the dominant term of C(n) will be $m/2$ ($\approx n^2/2$).

3.3. Prim's Shortest Spanning Tree Algorithm

The similar phenomena have been observed in the second computing experiment on the Prim's shortest spanning tree algorithm for complete graphs (where $l(u,v)=l(v,u)$ for any u,v). The Prim's algorithm is obtained by deleting two D[w]'s in line 8 in the description of the Dijkstra's algorithm. This algorithm is also optimal for complete graphs in the worst case [4].

See Table 3 for the counting results in the second experiment, which corresponds to Table 1 in the first experiment. This result seems to show almost the same behaviors for n as the result on the Dijkstra's algorithm for the uniform distribution. On more detailed examination of the computing results, the number of updating times in the Prim's case is a little (by about 0.6n) larger than that in the Dijkstra's case. The quite similar phenomena to Figure 2 have been observed. The plotted curves are almost identical to those in Figure 2 for small s for the same n's. The observed results in the second experiment are summarized as follows:

"The probability of updating for each vertex in V-P in the s-th UPDATE step may be well approximated to be 1/s for complete graphs. Hence the expected total number of updating times will be bounded by $n \log_e n$."

Note that in this case the experimental results hold not only for the uniform distribution but also for other types of distribution (3.4).

The analogous considerations based on those counting results may be applied to the efficient implementation of the Prim's algorithm.

3.4. Various Types of Distribution

In the third experiment several other types of distribution of input data have been considered. They are exponential, Erlang and normal. More specifically, those distributions have been generated by the following formulas:

$X=-(1/4)\log_e U$, for exponential

$Y=-(1/2)\sum_{i=1}^{k} \log_e U_i$, for Erlang with parameter k

$Z=\sum_{i=1}^{12} U_i$, for normal,

where U and U_i are chosen from the uniform distribution in [0,1). Note that the Erlang distribution with k=1 is identical to be

exponential.

The total numbers of updating times for such distributions have been counted in just the same manner as in the former two experiments. See Figure 4 for the counting results in the Dijkstra's algorithm. For the exponential case, the values are average out of 5 trials, while for the Erlang and normal cases, they are counted once for each.

As an extreme case suppose the distribution is constant, i.e., every edge has been weighted, say, 0.5. Obviously no updating will occur throughout the algorithm except the first initializing UPDATE step. This extreme case, along with the case of uniform distribution, suggests the following result observed in the third experiment:

> "The uniform distribution causes the expected number of updating times greater than other types of distribution, except the exponential distribution which shows the same behaviors as the uniform distribution."

Hence the proposed implementation technique for the uniform distribution seems to be effectively applicable to other types of distribution in order to bound the expected number of comparisons.

For the Prim's algorithm, those types of distribution have shown the same results as the uniform distribution, as reported in 3.3.

4. Intuitive Elucidation of the Results

We shall give an intuitive elucidation of the underlying grounds for our experimental results. In order to see asymptotic behaviors of the algorithms, the number of updating times will be considered only for relatively small s for sufficiently large n.

Suppose the following two conditions:

(a) for each vertex v in V , any edge in the set of incoming edges to v may have the properly minimum weight with an equal probability,

(b) the value of D[w] in line 8 is negligibly small.

Then p(n,s) will be "roughly" equal to 1/s under those conditions.

In most applications we can generally assume that condition (a) holds. In the Prim's algorithm condition (b) is always true, since D[w]=0. Hence the independence of several types of distribution considered in the Prim's case will be derived. In the Dijkstra's algorithm it is reasonable to guess that condition (b) holds for the uniform distribution. Assume that the initial weights of outgoing edges from any vertex are i/n (for i=1,2,...,n-1) as a very simplified model. Then it is easy to see that D[w] in the s-th step has a value almost equal to $(\log_2 s)/n$ for small s. For sufficiently large n this value may be neglected for small s. In fact, the value of D[w] in the s-th step has been demonstrated to be approximately $(\log_2 s)/n$ in the computing experiment. Note that the expected shortest distance may be conjectured to be bounded by $(\log_2 n)/n$ for the uniform distribution between 0 and 1.

As already pointed out, for other types of distribution considered here, except the exponential one, the probability of updating will be smaller, because the previously determined value of D[w] can not be neglected.

5. Conclusions

We have shown some expected behaviors of two algorithms for finding shortest paths and shortest spanning trees in complete graphs, and proposed their expectedly efficient implementations.† Those results will be useful for practical applications of the algorithms.

For the completeness of our study more rigorous treatments of our results should be given. The mathematical analysis based on the elementary probabilistic model has been partly elaborated by the first author, collaborated with Hajime Machida, which will be reported elsewhere.

---

† The present paper is based on the results obtained during the work of the second author [10], supervised by the first author. The second author is now with Musashino Electrical Communication Laboratory, NTT, Musashino, Tokyo 180, Japan.

References

[1] Dijkstra, E.W., "A Note on Two Problems in Connexion with Graphs," Num. Math., 1 (1959), 269-271.

[2] Prim, R.C., "Shortest Connection Networks and Some Generalizations," BSTJ (1957), 1389-1401.

[3] Aho, A., J. Hopcroft and J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley (1974).

[4] Spira, P.M. and A. Pan, "On Finding and Updating Spanning Trees and Shortest Paths," SIAM J. Comp., 4 (1975), 375-380.

[5] Johnson, E., "On Shortest Paths and Sorting," Proc. ACM 25th Annual Conf. (1972), 510-517.

[6] Johnson, D., "Algorithms for Shortest Paths," Ph.D. Thesis, Cornell Univ. (1973).

[7] Johnson, D., "Efficient Algorithms for Shortest Paths in Sparse Networks," JACM, 24 (1977), 1-13.

[8] Noshita, K., "On Data Structures for Manipulating Graphs and a New Efficient Program for the Dijkstra's Method for Shortest Path Problems," Third Symposium on Theory of Programs, Kyoto Univ. (1973).

[9] Knuth, D.E., *The Art of Computer Programming, Vol. 2 (Semi-numerical Algorithms)*, Addison-Wesley (1969).

[10] Masuda, E., "A Study on the Computational Efficiency of Shortest Path Algorithms," Master Thesis, Univ. Electro-Communications (Feb. 1977), 108 pp.

n	256	512	1024	2048
Updating	1317	2994	6654	14717
	(18)	(38)	(59)	(109)
$\sum_{s=1}^{n-1}(n-s)/s$	1311	2977	6663	14746

Table 1. The Total Number of Updating Times for the Uniform Distribution in the Dijkstra's Algorithm

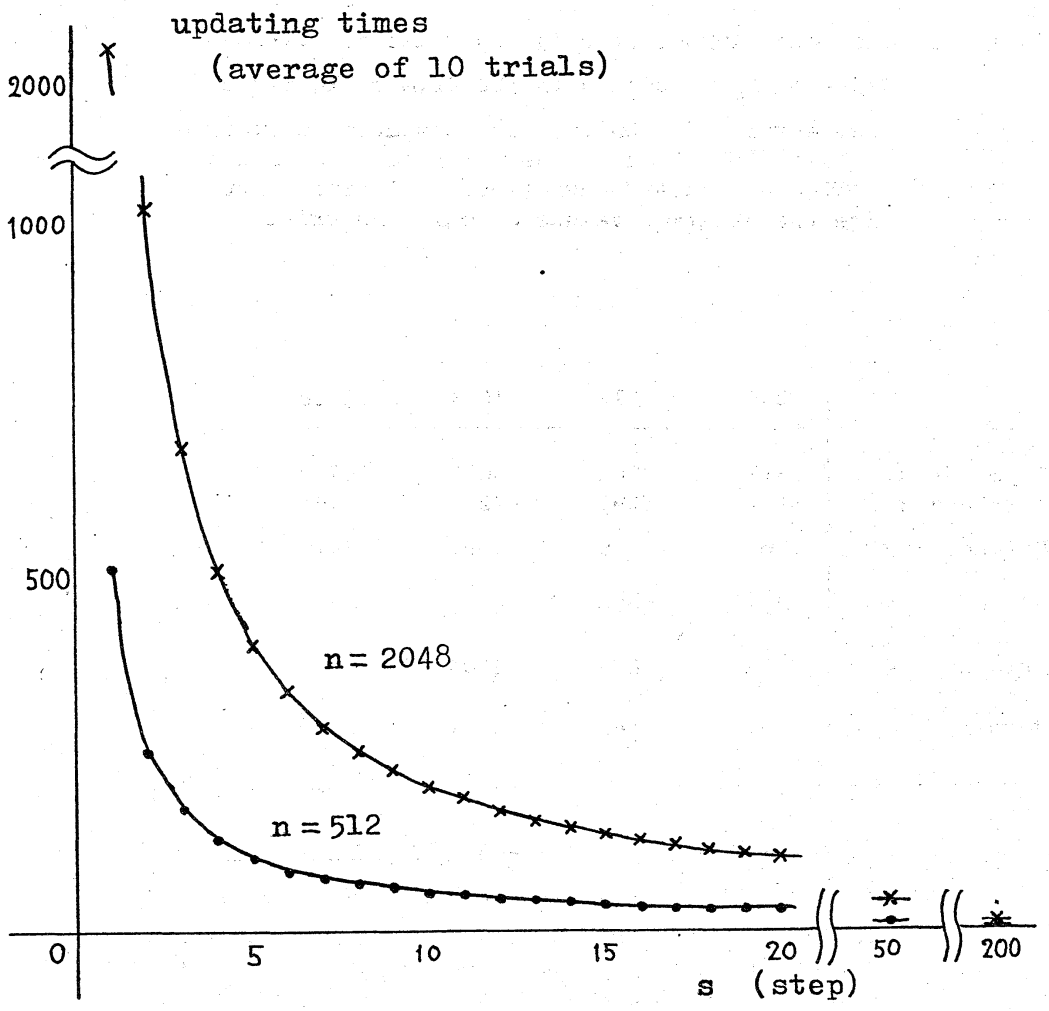

Fig. 2. Stepwise updating times for n=512 and 2048

n	256	512	1024	2048
Uniform	1473 (26)	3323 (48)	7296 (64)	15906 (76)
Exponential (≡Erlang k=1)	1445	3322	7324	16145
Erlang k=2	1450	3290	7353	16047
Erlang k=3	1443	3295	7193	16037
Erlang k=4	1449	3330	7333	15981
Normal	1465	3366	7340	16418

Table 3. The Total Number of Updating Times for Various Types of Distribution in the Prim's Algorithm

(The entries for Uniform are computed as average values of 10 trials. The parenthesized values indicate standard deviations. All other entries are the observed values of only one trial.)

n	256	512	1024	2048
Exponential (≡Erlang k=1)	1311 (10)	2984 (29)	6698 (29)	14770 (99)
Erlang k=2	1068	2419	5592	12416
Erlang k=3	892	1979	4504	10326
Erlang k=4	832	1824	4104	9225
Normal	262	546	1151	2439

Table 4. The Total Number of Updating Times for Other Types of Distribution in the Dijkstra's algorithm

(The entries for Exponential are computed as average values of 5 trials. The parenthesized values indicate standard deviations. All other entries are the observed values of only one trial.)